DSDISP
Version 3.0

Nakul Chitnis: nakul@math.arizona.edu
James Hyman: jh@lanl.gov
Juan Restrepo: restrepo@math.arizona.edu
Jia Li: li@math.uah.edu

June 8, 2004

Contents

1 Model

Abstract

In recent years, infectious diseases such as AIDS have greatly grown in prominence. Also, with the emergence of new diseases like SARS, the sporadic outbursts of ebola, plague and Creutzfeldt-Jakob disease (CJD) and the continuing bioterrorist threat of small pox, we require a better understanding of the spread of infectious diseases. A common way to study epidemics is to divide the population into three main groups — Susceptible, Infected, and Recovered (known as the SIR model). The entire population can also be differentiated according to demographic characteristics and the infected group can then be subdivided according to the stage of infection. Here we present a combined model that contains such a demographically differentiated population and allows each demographic group to pass through multiple stages of infection. This model is general enough to allow for the simulation of different diseases through populations divided according to various demographic categories, such as age, behavioural risk or income level. We have written a program in MATLAB to simulate this model and this program is available for use.

1.1 Theoretical Development

1.1.1 Introduction

To effectively study the spread of infectious diseases, we divide the population into three main stages — susceptible (S), infected, (I) and recovered (R). We then divide these stages into various demographic groups. This allows us to take into account different attributes of people such as general susceptibility, infectivity, number of contacts, rate of disease progression, death rate and so on. The possible demographic categories include age, income level, occupation, number of contacts, etc. We then further subdivide the infected groups into various infection stages. This allows us to take into account varying infectivity and death rates at different stages of the diseases. Although not yet included in the model, it would also be possible to model behaviour changes with different infection stages.

People enter the demographically differentiated susceptible groups at different rates through birth and migration. As they get infected, at a rate dependent on that demographic group, they enter the first infection stage. Then, depending on the rates of disease progression, they proceed through the remaining infection stages, before entering the recovered population. From any stage, they leave the model population at a rate derived from natural death and migration.

This is a good model for viral diseases that confer permanent immunity after a person has recovered. It is also possible to expand this model, with user-defined modules, to include diseases with temporary immunity.

1.1.2 Model Definition

DSDISP stands for Differential Susceptibility Differential Infectivity Staged-Progression. As described in the introduction, we are demographically dividing the population into different groups that progress through various infection stages. For this model, we will consider n demographic groups and m infection stages. See Figure 1 for the basic layout of the model. See (1) for the dynamics of the model and Table 1 for a description of the parameters used in this model. All parameters except λ are constant in time. We will define λ in §1.1.3.

In the current version of the model, people cannot move between demographic groups. Once someone enters a certain susceptible group in Figure 1, (s)he may only move horizontally through the infection stages of that demographic group. The groups in this model may therefore not be used to represent locations when the model includes migration between the locations. The groups can be used to represent locations only if people always return to their home location. Also, if the groups are used to represent age, the lifespan of the disease must be significantly shorter than the size of the age groups.

Eventually, we would like to add the ability to move between demographic categories. We would also like to include multiple strains of disease in a future version of the model.

$$\frac{dS_i(t)}{dt} = \Lambda_i - (\mu_{iS} + \lambda_i(t))S_i(t) \tag{1a}$$

$$\frac{dI_{i1}(t)}{dt} = \lambda_i(t)S_i(t) - (\gamma_{i1} + \mu_{i1})I_{i1}(t) \tag{1b}$$

$$\frac{dI_{ij}(t)}{dt} = \gamma_{i,j-1}I_{i,j-1}(t) - (\gamma_{ij} + \mu_{ij})I_{ij}(t) \qquad \text{for } 2 \le j \le m \tag{1c}$$

$$\frac{dR_i(t)}{dt} = \gamma_{im}I_{im}(t) - \mu_{iR}R_i(t) \tag{1d}$$

for $1 \le i \le n$.

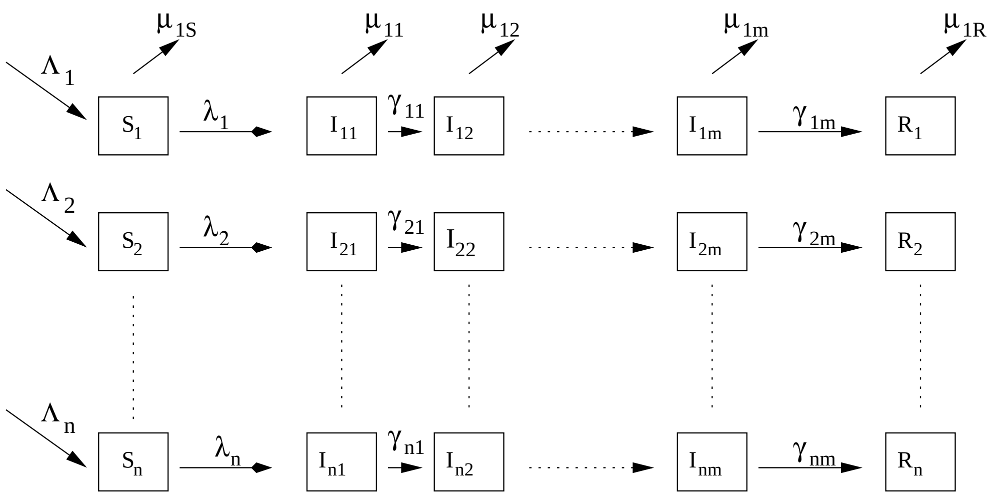

Figure 1: Schematic for the DSDISP model with n demographic groups and m infection stages. People enter different susceptible groups through birth or migration at different rates dependent on the demographic characteristics of that group. They get infected at a rate, λ, and then progress through various infection stages at rates of disease progression, γ, before entering the recovered state. They leave the population at a rate, μ, through death and migration.

1.1.3 Definition of the rate of infection (λ)

The core of (1) lies in the definition of λ_i. We define λ_i as the rate at which the susceptible population in demographic group i gets infected and progresses to stage I_{i1}. We calculate this as the sum of the rate of disease transmission from each infected subgroup, I_{kj} for $1 \le k \le n$ and $1 \le j \le m$, to the susceptible group, S_i, in (2). This means that a susceptible person in group i can get infected by an infected person in any group or infection stage.

$$\lambda_i(t) = \sum_{k=1}^{n} \sum_{j=1}^{m} \lambda_{ikj}(t) \tag{2}$$

Here, λ_{ikj} is the rate of disease transmission from the infected people in subgroup I_{kj} to the susceptibles in S_i. We calculate this in (3) as a product of the number of contacts, per unit time, that each individual in group i has with demographic group k; the probability of transmission per contact between someone in group I_{kj} and group S_i; and the probability that out of all people in demographic group k, the contact is in infection stage j. See Table 2 for a summary of the parameters used in this definition.

$S_i(t)$:	The number of susceptibles in demographic group i. Units: People.
$I_{ij}(t)$:	The number of infecteds in demographic group i and infection stage j. Units: People.
$R_i(t)$:	The number of recovereds in demographic group i. Units: People.
$\lambda_i(t)$:	Rate at which susceptibles in demographic group i get infected. Units: Time^{-1}.
Λ_i:	The inward migration rate for demographic group i. Units: People/Time.
γ_{ij}:	The rate of disease progression for a person in demographic group i and infection stage j. Units: Time^{-1}.
μ_{iS}:	$= \mu_{im} + \mu_{id}$. The total loss of population from group S_i. Units: Time^{-1}.
μ_{ij}:	$= \delta_{ij} + \mu_{im} + \mu_{id}$. The total loss of population from group I_{ij}. Units: Time^{-1}.
μ_{iR}:	$= \mu_{im} + \mu_{id}$. The total loss of population from group R_i. Units: Time^{-1}.
μ_{im}:	The natural migration rate out of the population for demographic group i. Units: Time^{-1}.
μ_{id}:	The natural death rate for demographic group i. Units: Time^{-1}.
δ_{ij}:	The disease induced death rate for demographic group i in infectious stage j. Units: Time^{-1}.

Table 1: Summary of the parameters in the DSDISP model. All parameters are nonnegative.

$$\lambda_{ikj} = \begin{pmatrix} \text{Number of} \\ \text{Contacts per} \\ \text{Unit Time} \end{pmatrix} \begin{pmatrix} \text{Probability of} \\ \text{Transmission} \\ \text{per Contact} \end{pmatrix} \begin{pmatrix} \text{Probability} \\ \text{Contact is} \\ \text{Infected} \end{pmatrix}$$

$$\lambda_{ikj}(t) = (r_{ik}(t))\,(\alpha_i \beta_{kj}) \left(\frac{I_{kj}(t)}{N_k(t)} \right) \tag{3}$$

The term, (r_{ik}), gives the average number of contacts made by one person in demographic group i with all people in demographic group k, as defined in (5). We explain this term in more detail in §1.1.4. The transmissibility, $(\alpha_i \beta_{kj})$, is the probability that someone in I_{kj} infects a person in S_i, given that there is a contact between I_{kj} and S_i. It is the product of the susceptibility of group S_i, α_i, and the infectivity of subgroup I_{kj}, β_{kj}. The term, $(I_{kj}(t)/N_k(t))$, gives the fraction of individuals in group k who are in infected stage I_{kj}. The denominator, $N_k(t)$, is the total population size in demographic group k as in (4). The product, $r_{ik} I_{kj}/N_k$, gives the average number of contacts between someone in demographic group i and the infected people in subgroup I_{kj}. Multiplying that by the probability of transmission gives the force of infection from subgroup I_{kj} to the susceptibles in demographic group i. Summing over all the infection stages gives the force of infection from all infecteds to the susceptibles in group i. Multiplying this quantity by the number of susceptibles as in (1) gives the rate of change of new infecteds in demographic group i.

$$N_k(t) = S_k(t) + \sum_{j=1}^{m} I_{kj}(t) + R_k(t) \tag{4}$$

$$r_{ik}(t) = c_i p_{ik} \frac{c_k N_k(t)}{\sum_{l=1}^{n} c_l N_l(t)} \qquad \text{with} \qquad p_{ik} = q_{ik} q_{ki} \tag{5}$$

$\lambda_{ikj}(t)$:	The rate of disease transmission from I_{kj} to S_i. Units: Time^{-1}.
α_i:	$\in [0,1]$. The susceptibility of a person in S_i. Units: 1.
β_{kj}:	$\in [0,1]$. The infectivity of a person in I_{kj}. Units: 1.
$r_{ik}(t)$:	The average number of contacts each person in demographic group i has with group k per unit time. Units: Time^{-1}.
c_i:	The preferred number of contacts (per person per time) for people in demographic group i. Units: Time^{-1}.
p_{ik}:	The probability (or preference) for a contact between group i and group k. Units: 1.
q_{ik}:	The desirability of a contact with group k for someone in group i. This can also be thought of as the acceptability, for contact, of group i for group k. Units: 1.

Table 2: Summary of parameters in the definition of λ. All parameters are nonnegative.

Dimensional Analysis:
Let $[x]$ denote the units of x for some variable x.

$$[\alpha_i] = 1$$
$$[\beta_{kj}] = 1$$
$$[r_{ik}(t)] = \frac{\text{Contacts}}{\text{Time} \cdot \text{Person}} = \frac{1}{\text{Time}}$$
$$[p_{ik}] = 1$$
$$\left[\frac{I_{kj}(t)}{S_k(t) + \sum_{j=1}^{m} I_{kj}(t) + R_k(t)} \right] = 1$$

Therefore,

$$[\lambda_{ikj}(t)] = \frac{1}{\text{Time.}}$$

1.1.4 Definition of the mixing

The pattern of contacts between different groups plays an essential role in determining the spread of disease, especially in sexually transmitted diseases. We assume people in each group behave the same way when selecting a partner, but have biases between groups. In other words, mixing within each group is assumed to be homogeneous but there is heterogeneous mixing among the groups.

This mixing between groups is one of the most important factors in modeling diseases. For sexually transmitted diseases, it depends on the desirability of an active individual, the acceptability of his/her potential contacts, and the availability of these potential contacts.

Let q_{ik} be the desirability of people in group i to have a contact from group k; that is, q_{ik} is the fraction of people in group k with whom an individual in group i desires forming a contact. Thus q_{ik} describes the desirability of people in group i to have a contact from group k. It is also the acceptability of people in group k to people in group i.

Under the condition that enough potential partners are available, the probability p_{ik} that a partnership forms between individuals from group i and group k, is the product of the availability of group i for group k, q_{ki}, and the desirability of group i for group k, q_{ik}, as in (5).

Note that we can also alternatively define p_{ik} as the preference for a contact between group i and group k. With this alternative definition, the p_{ik}'s are no longer restricted to being less than or equal to 1.

We define c_i to be the preferred number of social contacts per unit time for a person in group i. The probability that a contact is with a person from group k is $c_k N_k/(\sum_l c_l N_l)$ where N_k is the total population size of group k defined in (4). This also characterizes the availability of contacts with partners in group k. Hence, the probability of a partnership forming between individuals from group i and group k is $p_{ik} c_k N_k/(\sum_l c_l N_l)$ (Again, if we think of p_{ik} as a preference then this now becomes a preference of forming partnerships.)

The desirability matrix need not be symmetric (i.e. $q_{ik} \neq q_{ki}$, when $i \neq k$), but the probability of a partnership forming is symmetric since $p_{ik} = q_{ik} q_{ki}$ implies $p_{ik} = p_{ki}$. Also, we note that there is no constraint on $\sum_k q_{ik}$, which may be less than or greater than one.

Two special cases of the model (1) with the infection rate, (2) and (3), are the restricted mixing model when $q_{ik} = 0$ (hence $p_{ik} = 0$, $i \neq k$) and the proportional mixing model when $q_{ik} = 1$, for $\{i, k\} = 1, \cdots, n$.

We denote the number of contacts per unit time of people in group i with people in group k by T_{ik}. The number of contacts with people in group i that people in group k have is also T_{ik}, that is $T_{ik} = T_{ki}$. These are the balance constraints that need to be satisfied at all times. In multi-group models where an attempt is made to directly control the number of partnerships formed between groups, these balance conditions usually are artificially enforced. However, in the selective mixing model, the balance constraint

$$T_{ik} = p_{ik} \frac{c_k N_k}{\sum_l c_l N_l} c_i N_i = p_{ki} \frac{c_i N_i}{\sum_l c_l N_l} c_k N_k = T_{ki} \tag{6}$$

is automatically satisfied. Thus, by using the acceptability q_{ik} or desirability q_{ki} of an individual from group i to an individual from group k as the primary control variable in these models (instead of the number of partners an individual from group i desires from group k), the balance constraints become a natural consequence of the model, rather than an artificially imposed constraint.

The number of contacts per individual per unit time in many multi-group models is assumed to be constant. When all q_{ik}'s equal one (proportional mixing), this is also true for the selective mixing model. However, if the mixing is biased, the actual number of contacts, denoted by r_i, for the selective mixing model will vary in time depending on the combination of desirability, acceptability, and availability.

Define $P(i)$ as the probability that an individual in group i finds a partner from any group. The actual number of contacts per person in group i,

$$r_i = c_i P(i) = c_i \left(\sum_{k=1}^{n} p_{ik} \frac{c_k N_k}{\sum_l c_l N_l} \right), \tag{7}$$

reaches its maximum c_i only for the proportional mixing, where $p_{ik} \equiv 1$ (i.e. everyone is acceptable as a partner). Remember that the average number of contacts per person (7) is the sum over the contacts with each demographic group, (5).

If the mixing is biased, the acceptability and the availability of partners must be taken into consideration and a limitation may occur. Then $p_{ik} \leq 1$, and hence $r_i \leq c_i$.

However, we think of the p_{ik}'s as preferences, then it is possible for the actual number of contacts per person, r_i, to be greater than the preferred number of contacts, c_i.

1.1.5 Linear Analysis

The concept of a threshold condition is very important in epidemiology. The basic reproductive number, R_0, is the expected number of secondary infections from one infected person in a fully susceptible population during the duration of the entire infectious period. We are currently working on an analytic formulation for R_0.

However, we can evaluate the stability of the disease-free equilibrium through linear analysis. The disease-free equilibrium point of (1) is

$$S_i^* = \Lambda_i / \mu_{iS} \tag{8a}$$
$$I_{ij}^* = 0 \tag{8b}$$
$$R_i^* = 0. \tag{8c}$$

with $N_i^* = S_i^* + \sum_{j=1}^{m} I_{ij}^* + R_i^*$. There is a problem here if we have no inward or outward migration. We look at the four possible cases, for a given group, i, below.

1. If both Λ_i and μ_{iS} are nonzero, we can use (8) above without any problems.

2. If both, Λ_i and μ_{iS} are zero, the entire subspace, $I_{ij}^* = 0$ with S_i^* and R_i^* arbitrary, is a continuum of fixed points. In this case, we define the disease-free equilibrium to be $S_i^* = N_i(0)$; $I_{ij}^* = 0$ (which is the only mathematical requirement for the fixed point); and $R_i^* = 0$. We perform

our linear analysis around the point where there is no recovered population in that group and the susceptible population is equal to the input initial total population of that group.

3. If $\Lambda_i = 0$ and $\mu_{iS} \neq 0$, the disease-free equilibrium is $S_i^* = 0$; $I_{ij}^* = 0$; and $R_i^* = 0$.

4. If $\Lambda_i \neq 0$ and $\mu_{iS} = 0$, we have no equilibrium point because we have inflow but no outflow. We do not perform any linear analysis.

It is possible in a multi-group model for a number of the cases above to simultaneously occur. If 4 is true for any of the groups, there is no disease-free equilibrium and we do not conduct any linear analysis. If 2 or 3 are true for any of the groups, we pick the coordinates of the disease-free equilibrium for those groups as described above; and continue with the rest as we would with 1[1].

We calculate the Jacobian of the infecteds from our original system of equations (1). The first n equations correspond to the demographic groups in the first infection stage. The next n equations correspond to the demographic groups in the second infection stage and so on. The resulting Jacobian is $(m*n) \times (m*n)$. For our equations, the first n rows of the Jacobian are dense; while the remaining $(m-1)*n$ rows are sparse with entries in the main diagonal and the off-diagonal n rows across. The format of the Jacobian for a system with 2 demographic groups ($n = 2$) and 3 infection stages ($m = 3$) can be seen in (9). All other entries are 0.

$$Df = \begin{pmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & & * & & & \\ & * & & * & & \\ & & * & & * & \\ & & & * & & * \end{pmatrix} \tag{9}$$

As the actual entries of the Jacobian are too large to fit in one equation, we write them separately in (10)–(13). Let $\bar{\imath}$ and $\bar{k}$ correspond to the demographic group; and $\bar{\jmath}$ correspond to the infection stage. We label the first n rows by $\bar{\imath}$. We label all columns for the first n rows by $n * (\bar{\jmath} - 1) + \bar{k}$. (We remark here that $\bar{\jmath} = 1$ corresponds to the first n columns; $\bar{\jmath} = 2$ corresponds to the next n columns and so on.) Since we only have 2 nonzero diagonals in the remaining rows we can label them using only $\bar{k}$ and $\bar{\jmath}$. The main diagonal elements are labelled by $(n*(\bar{\jmath}-1)+\bar{k}, n*(\bar{\jmath}-1)+\bar{k})$. The off-diagonal elements are labelled by $(n*(\bar{\jmath}-1)+\bar{k}, n*(\bar{\jmath}-2)+\bar{k})$.

The terms in the Jacobian are as follows:

- For the main diagonal in the first n rows ($1 \leq \bar{\imath} = \bar{k} \leq n$ and $\bar{\jmath} = 1$):

[1]The model breaks down if 3 is true for all the groups.

$Df(\bar{\imath}, n*(\bar{\jmath}-1)+\bar{k}) =$

$$\left(\frac{c_{\bar{\imath}}c_{\bar{k}}p_{\bar{\imath}\bar{k}}\alpha_{\bar{\imath}}\beta_{\bar{k}\bar{\jmath}}}{\sum_{l=1}^{n} c_l N_l^*} - \sum_{k=1}^{n}\sum_{j=1}^{m} \frac{c_{\bar{\imath}}c_{\bar{k}}c_k p_{\bar{\imath}k}\alpha_{\bar{\imath}}\beta_{kj} I_{kj}^*}{\left(\sum_{l=1}^{n} c_l N_l^*\right)^2}\right) S_{\bar{\imath}}^* - \left(\gamma_{\bar{k}\bar{\jmath}} + \mu_{\bar{k}\bar{\jmath}}\right) \tag{10}$$

- For all other elements in the first n rows ($1 \le \bar{\imath}, \bar{k} \le n$, $1 \le \bar{\jmath} \le m$ and $\bar{\imath} \ne n*(\bar{\jmath}-1)+\bar{k}$):
  $Df(\bar{\imath}, n*(\bar{\jmath}-1)+\bar{k}) =$

$$\left(\frac{c_{\bar{\imath}}c_{\bar{k}}p_{\bar{\imath}\bar{k}}\alpha_{\bar{\imath}}\beta_{\bar{k}\bar{\jmath}}}{\sum_{l=1}^{n} c_l N_l^*} - \sum_{k=1}^{n}\sum_{j=1}^{m} \frac{c_{\bar{\imath}}c_{\bar{k}}c_k p_{\bar{\imath}k}\alpha_{\bar{\imath}}\beta_{kj} I_{kj}^*}{\left(\sum_{l=1}^{n} c_l N_l^*\right)^2}\right) S_{\bar{\imath}}^* \tag{11}$$

- For the main diagonal in the remaining rows ($1 \le \bar{k} \le n$ and $2 \le \bar{\jmath} \le m$):
  $Df(n*(\bar{\jmath}-1)+\bar{k}, n*(\bar{\jmath}-1)+\bar{k}) =$

$$-\left(\gamma_{\bar{k}\bar{\jmath}} + \mu_{\bar{k}\bar{\jmath}}\right) \tag{12}$$

- For the off-diagonal in the remaining rows ($1 \le \bar{k} \le n$ and $2 \le \bar{\jmath} \le m$):
  $Df(n*(\bar{\jmath}-1)+\bar{k}, n*(\bar{\jmath}-2)+\bar{k}) =$

$$\gamma_{\bar{k},\bar{\jmath}-1} \tag{13}$$

If all the eigenvalues of this Jacobian, evaluated at the disease-free equilibrium point, are in the negative half-plane, then we can conclude that the disease-free equilibrium is locally asymptotically stable as small changes in the infected population will exponentially decay to zero.

We are also currently looking into ways of characterising the basic reproductive number as a distribution over the different demographic groups.

1.1.6 Initialization

An added complication in any model with infected subgroups is the dependence of the behavior and the timing of the transient solutions of the epidemic model on the initial distribution of the infected populations among the subgroups. Hyman et al. show in [2] that even when the total number of infected individuals is fixed, the timing of the epidemic can be shifted by large amounts of time by varying the distribution of the initial infected population. This important observation is often overlooked in simulation studies comparing multi-group models. We develop a robust, systematic procedure for determining the initial distribution of the infected population when $R_0 > 1$, based on the progression of a natural epidemic that sets the timing of different multi-group models and allows them to be quantitatively compared.

Let $f_{kj} = I_{kj}(0)/I(0)$. The total initial infected population,

$$I(0) = \sum_{k=1}^{n}\sum_{j=1}^{m} I_{kj}(0),$$

is fixed and the goal is to prescribe a robust procedure to define the fractions, f_{kj}, where $\sum_{k=1}^{n}\sum_{j=1}^{m} f_{kj} = 1$.

For models of the spread of infectious agents such as HIV, which have been spreading into populations that were originally free of this disease and thus have an initial background level of zero infections, we propose a procedure for defining the f_{kj} when $R_0 > 1$ based on the idea that in nature the epidemic would have spread into the population from only a few infected individuals. The epidemic then grows from this small seed until by the time a noticeable number of individuals are infected, a natural balance between the infected subgroups is reached (thus defining f_{kj}). When $R_0 > 1$, then the *numerical pre-initialization procedure* (NPP) is a simple approach to approximating the natural balance in the population that exists when an epidemic started in the past. In this procedure we distribute the initial infected population based on what it would be if a very small initial infected population was introduced in the distant past and grew to infect a given percentage of the population.

For example, to use the NPP simulation to distribute a 1% initial infection rate, the pre-initialization simulation is started with a much smaller infected population, say 0.01% of the population infected, and the equations are integrated until 1% of its population has become infected. This occurs at some time t^P (which depends on the distribution over the infected subgroups of the 0.01% pre-initialization infected population). At time t^P we stop the simulation and use the evolved distribution of the infected population from the pre-initialization simulation to define the relative fraction of the initial infected population in each group for the main simulation. Thus $f_{kj} = I_{kj}^P(t^P)/I^P(t^P)$, where the superscript P denotes the solution and ending time of the pre-initialization simulation. In numerical experiments we observed that the resulting initial conditions are almost independent of the distribution used in the pre-initialization simulation.

1.2 References

References

[1] Hyman, J. M. & Li, J. (1997). Disease Transmission Models with Biased Partnership Selection. *Applied Numerical Mathematics* **24,** 379–392.

[2] Hyman, J. M., Li, J. & Stanley, E. A. (1999). The differential infectivity and staged progression models for the transmission of HIV. *Mathematical Biosciences* **155,** 77–109.

[3] Hyman, J. M., Li, J. & Stanley, E. A. (2001). The Initialization and Sensitivity of Multigroup Models for the Transmission of HIV. *J. theor. Biol.* **208,** 227–249.

[4] Li, J. (2002). Multi-Group Epidemiological Models.

2 Code Description

2.1 Availability

On the website `http://www.math.arizona.edu/~nakul/dsdisp.html`, there is a link to the DSDISP model. This link contains this documentation and the code.

You can download the code here — for Windows and for Linux. There is documentation in html and pdf form.

2.2 Conditions for Use

This code is freely distributed for **non-commercial** use.

- Non-commercial uses: In exchange for its use the authors REQUIRE proper acknowledgement in electronic form (when used as part of a larger code) and/or in printed form (when used in projects that lead to any type of publication).
- Commercial uses:
  - REQUIRE proper acknowledgement in electronic form (when used as part of a larger code) and/or in printed form (when used in projects that lead to any type of publication). Code modifications REQUIRE the consent of the authors.
  - AND REQUIRE permission from the authors.

The code is distributed "as is". The authors will not assume any legal responsibilities arising from problems with this package.

This code is maintained by the authors. We welcome any suggestions for improvements and will gingerly and promptly take care of any bugs. Suggestions for modifications will be given serious consideration.

2.3 Code Documentation

- `dsdisp.html` (this document).
- `dsdisp.pdf` (this document).
- Help files on MATLAB scripts and functions.

2.4 To Obtain DSDISP code

2.4.1 Linux/UNIX

Download `dsmodel.tar.gz` from `http://www.math.arizona.edu/~nakul/dsdisp.html`.

2.4.2 Windows

Download `dsmodelfiles.zip` from `http://www.math.arizona.edu/~nakul/dsdisp.html`.

2.5 Code Installation

2.5.1 Linux/UNIX

After you have downloaded `dsmodel.tar.gz`, in the directory where you would like to create the subdirectory `dsmodel/.`, type the following:

1. `gunzip dsmodel.tar.gz`
2. `tar xvf dsmodel.tar`

That should create the directory `dsmodel` and all its subdirectories.

2.5.2 Windows

After you have downloaded `dsmodelfiles.zip`, you may use Winzip to extract the files into a directory of your choice. The extraction will contain the subdirectory `dsmodel`.

2.6 System/software Requirements

- There are no system requirements.
- Platforms: any LINUX/UNIX flavor or Windows[2]. We believe it will also work on a Macintosh although we have not tested that.
- MATLAB (version 6.1.0.450 (R12.1)). Although we believe the code should work on some earlier versions of MATLAB[3].

3 Code Operation

3.1 Code Structure

The directory `dsmodel` has five subdirectories:

- `code`
- `input`
- `output`
- `docs`
- `aux`.

[2]We have tested the code for Windows XP.

[3]There may be problems on earlier versions of MATLAB for Windows because the code requires a distinction between lowercase letters and capital letters.

1. The directory `code` contains all the scripts and functions that make up this code. The main driver, `dsdisp.m`, and `plotfig.m` are scripts while the rest of the subroutines are functions. This directory has three subdirectories for three MATLAB classes. The structure of these classes is explained in §3.2. The subdirectories are:
   - `@xtype`
   - `@tautype`
   - `@xtautype`.
2. The `input` directory contains the input files:
   - `sp.in`
   - `SIfile`
   - `XTfile`
   - `MMfile`.

   These files are described in §3.4. This directory also contains a number of subdirectories that contain examples of input files that may be used for sample runs. See §3.7 for more information on how to use these for sample runs.
3. The `output` directory contains the files created by the code where the data is saved. See §3.6.2 for the details of these files.
4. The `docs` directory contains this manual in pdf and html format.
5. The `aux` directory contains auxiliary files such as Maple scripts used in creating this code.

3.2 Data Structure

The data structure is broken into 4 primary classes:

- Scalars
- Variables dependent on the demographic group (`xtype` classes).
- Variables dependent on the infection stage (`tautype` classes).
- Variables depending on both the demographic group and the infection stage (`xtautype` classes).

The `xtype`, `tautype` and `xtautype` variables have been created as MATLAB classes. To see the methods associated with each class, type in

```
> methods <classname>
```

For example,

```
> methods xtype
```

will display the methods associated with the `xtype` class. You can then type

```
> help <classname>/<methodname>
```

for help on a particular method. For example,

```
> help xtype/printx
```

will show the help file for printing the values of an `xtype` class. We explain the format of each class in some detail below.

1. The `xtype` classes store data or parameters that are dependent on the demographic group. Examples of these include the susceptibility of the susceptible population, the inward migration rate and the susceptible and recovered population. To enable these variables to represent either absolute numbers or densities we represent the data in midpoint and endpoint format. The values at the midpoint can be used to represent the actual numbers or they can be assumed to represent densities and extrapolated to give values for the endpoints. We will use x to represent the demographic divisions. If, for example, we wish to use age as our demographic category, the values of x would represent the divisions of the different age groups. The bounds of the age bins would serve as the endpoints and these can then be used to find the midpoints of the age bins. An example of dividing a population into 6 age bins is (0, 5], (5, 12], (12, 18], (18, 22], (22, 55], (55, 100]. The corresponding x endpoints then are 0, 5, 12, 18, 22, 55 and 100. The corresponding midpoints are 2.5, 8.5, 15, 20, 38.5 and 77.5. The current version of the code only accepts endpoints for x and interpolates these to find the midpoints. Even if the demographic category is non-numeric (eg. gender or occupation), the code will still require x endpoints. For 4 demographic groups, the endpoints could simply be 1, 2, 3, 4 and 5. Note that the number of x midpoints is equal to the number of demographic groups and the number of x endpoints is one greater than the number of groups.

   The `xtype` classes store these endpoint and midpoint values for x and store corresponding values for the variable. The code currently only deals with actual numbers (not densities) so the values of the variable at the midpoints corresponds to the actual number of that variable for that demographic group. In the above example of age groups, if the value for the first midpoint (2.5) for the recovered population is 240, then we can interpret that as 240 people in the recovered class between the ages of 0 and 5. The code now only accepts input data for the variable at the midpoints. It then extrapolates this to find the values of the variable at the endpoints. In this version of the code, we do not use the data at the endpoints. (In future versions of the code, we may also accept data at the endpoints and may work with distribution densities too.)

2. The structure of the `tautype` classes is similar to that of the `xtype` classes, except the variables are now a function of the infection stage, τ, instead of the demographic group, x. There are no significant differences between these two classes except that the number of midpoints now is equal to the number of infection stages; and the τ endpoints represent the time

frames of the different stages of the disease. We do not currently have any variables that are only dependent on the infection stage but we use this class of variables to create `xtautype` classes.

3. The `xtautype` store data or parameters that are dependent on both the demographic group and the infection stage. Examples of these include the infectivity of the people in the different infected subgroups and the number of infected people in each of these subgroups. They are created by combining the `xtype` and `tautype` classes, using the relative values of the different demographic groups and the infection stages. The sum of the data for the variable for both of these is then normalized to 1 and the two are raked together to create an `xtautype` class. The data is then multiplied by the value of the integral (specified in the input files — see §3.4.1).
   Raking: We write the `xtype` data as a normalized $(n \times 1)$ vector and the `tautype` data as a normalized $(1 \times m)$ vector. We then multiply the x data by the τ data to create a $(n \times m)$ matrix — the data for the `xtautype` variable. This matrix is then multiplied by the value of the integral.

3.3 Initial Conditions

The initial conditions are read in through input files. Nonzero values can only be specified for the initial susceptible and infected populations. The initial recovered population is always set to 0.

The model is run through an initialization procedure as described in §1.1.6. The pre-initialization infected population is read through input files as defined in §3.4. The code is then run until some threshold value of infecteds, I^P, is reached, at some time, t^P [4].

The time is then reset to 0 and the code run again with initial conditions defined by the state of the system at time t^P. The threshold value can be either set as the actual number of infecteds or as a percentage of total *pre-initialization* population. See §3.4 for details on how to set this threshold value.

At the time the threshold value is reached, the populations of all groups, S_i, I_{ij} and R_i, and the time are stored in `Sop`, `Iop`, `Rop` and `tronep`, respectively. Note that `Sop`, `Iop` and `Rop` are stored as classes. The time rate of change of the populations are stored in `Stop`, `Itop` and `Rtop`. These are also stored as classes. The the time the threshold is reached, t^P, is also stored in an output file as described in §3.6.2.

There is also a maximum set for initialization time, t^M. If the code does not reach the threshold value for the the infected population by t^M, it uses values in the different groups at time t^M as the new initial conditions, resets the time and continues. The code does display a warning.

If the pre-initialization infected population specified in the input files is greater than the given threshold, I^P, the code displays a warning, and does not

[4]The actual time when the infected population reaches the threshold value is calculated by interpolating the two closest values from the output of MATLAB's `ode45` command.

follow the pre-initialization procedure. It simply uses the given pre-initialization values as the initial conditions.

3.4 Input Files

3.4.1 `sp.in`

The main input file is `sp.in` contained in the directory `dsmodel/input/`. The parameters in this file is **order sensitive**. Changing the order of the parameters will result in an error that may or may not be displayed on the screen. The data from this file is read using the command `textread`. All variables that are stated to functions of x must have as many elements as the number of demographic groups. Similarly, all variables stated to be functions of τ must have as many elements as the number of infection stages[5]. The input parameters are:

`FlagData`:	A flag for the format of `SIfile`. Can be either '`FullMP`', '`FullEP`', '`SparseMP`' or '`SparseEP`'. Class: String. See §3.4.3 for details.
`SIfile`:	The name of the file where the initial population data is stored. This contains the susceptible and infected populations used for the pre-initialization procedure. The recovered population is automatically set to 0. Class: String. See §3.4.3 for a description of the format of this file. Note that if the file is in the directory `dsmodel/input/.`, the name will need to contain '`../input/.`' because the code is run from the directory `dsmodel/code/.`.
`XTfile`:	The name of the file where the data for the x and τ variables is stored. Class: String. See §3.4.2 for details of the format of this file.
`MMflag`:	A flag that determines the format of `MMfile`. This can be either '`MixingMatrix`', '`MixingMatrixPP`' or '`QCData`'. Class: String. See §3.4.4 for more details.
`MMfile`:	The name of the file where the mixing matrix is stored. Class: String. See §3.4.4 for details of the format of this file.
`Initflag`:	This is a string that can either be '`Percent`' or '`Number`'. This is used to determine the threshold value for the total infected population size at which the code re-initializes. If '`Percent`' is selected, then `Initpc` is used; if '`Number`' is selected, then `Init_pop` is used. Class: String.
`Init_pop`:	I^P. The number of infected people at which to re-initialize time when `Initflag` is set to '`Number`'. Class: Scalar. Units: People.

[5]It may be easiest to follow the format specified in the sample input files for `sp.in` as described in §3.7

`Initpc`: When `Initflag` is set to '`Percent`', the number of infected people at which time is re-initialized is this percent of the **pre-initialization** population (the total population specified in `SIfile`). Class: Scalar. Units: Percentage.

`inittime`: t^M. The maximum time to run until re-initialization. If the infected population does not reach the threshold by this time, the code displays a warning and continues. This value needs to be a multiple of `dtsave`. Class: Scalar. Units: Time.

`tfinal`: The final time (after re-initialization) to integrate the system to. This value needs to be a multiple of `dtsave`. Class: Scalar. Units: Time.

`dtsave`: The time interval at which data is saved. This value needs to be a factor of `tfinal` and `inittime`. If it is not a factor of either one of the two, the code will run, but will display a warning. A large value of `dtsave` will result in faster runs but coarser plots and coarsely saved data, while a small value will result in slower runs but finer plots and larger data sets. Class: Scalar. Units: Time.
Note: The value of `dtsave` will not affect the accuracy of the actual integration.

`IMR(x)`: Λ_i. The inward migration rate for demographic group i. Class: `xtype`. Units: People/Time.

`mud(x)`: μ_{id}. The natural death rate for demographic group i. Class: `xtype`. Units: Time^{-1}.

`mum(x)`: μ_{im}. The natural migration rate out of the population for demographic group i. Class: `xtype`. Units: Time^{-1}.

`alpha(x)`: $\alpha_i \in [0, 1]$. The susceptibility of a person in S_i. Class: `xtype`. Units: 1.

[Beta]:	**Note: [Beta] is not present in the input file.** The next 3 input parameters relate to this quantity. $\beta_{kj} \in [0,1]$. The infectivity of a person in I_{kj}. Class: `xtautype`. Units: 1.
`beta(x)`:	$\beta_{kj}(k)$. The relative infectivity of each demographic group. The sum of these parameters will be normalized to 1.
`beta(tau)`:	$\beta_{kj}(j)$. The relative infectivity of each infection stage. The sum of these parameters will be normalized to 1.
`beta (Value of Integral)`:	After normalized `beta(x)` and normalized `beta(tau)` have been raked, the resulting matrix will be multiplied by `beta (Value of Integral)`. This parameter is equal to $\sum_{k=1}^{n}\sum_{j=1}^{m}\beta_{kj}$. Care must be taken when this parameter is chosen to ensure that $\beta_{kj} \leq 1 \quad \forall\ k,j$. The code will **not** return a warning if this condition is violated.
[Gamma]:	**Note: [Gamma] is not present in the input file.** The next 3 input parameters relate to this quantity. γ_{ij}. The rate of disease progression for a person in demographic group i and infection stage j. Class: `xtautype`. Units: Time^{-1}.
`gamma(x)`:	The relative rates of disease progression for each demographic group. The sum of these parameters will be normalized to 1.
`gamma(tau)`:	The relative rates of disease progression for each infection stage. The sum of these parameters will be normalized to 1.
`gamma (Value of Integral)`:	After normalized `gamma(x)` and normalized `gamma(tau)` have been raked, the resulting matrix will be multiplied by `gamma (Value of Integral)`. This parameter is equal to $\sum_{i=1}^{n}\sum_{j=1}^{m}\gamma_{ij}$.

[DeltaI]:	**Note: [DeltaI] is not present in the input file.** The next 3 input parameters relate to this quantity. δ_{ij}. The disease-induced death rate for demographic group i in infectious stage j. Class: `xtautype`. Units: Time^{-1}.
`deltaI(x)`:	This is the relative death rate for each demographic group. The sum of these parameters will be normalized to 1. If they are all 0, no normalization will take place.
`deltaI(tau)`:	This is the relative death rate for each infection stage. The sum of these parameters will be normalized to 1. If they are all 0, no normalization will take place.
`deltaI (Value of Integral)`:	After normalized `deltaI(x)` and normalized `deltaI(tau)` have been raked, the resulting matrix will be multiplied by `deltaI (Value of Integral)`. This parameter is equal to $\sum_{i=1}^{n}\sum_{j=1}^{m}\delta_{ij}$.

`plotfigures`: Gives different options for plotting the results. §3.6.1 has more details. Class: Scalar.

- 0: Prints no plots.
- 1: Prints out separate plots for each demographic group. Each plot will show the susceptible population, the population in each infected stage, and the recovered population in that group.
- 2: Prints out separate plots for each demographic group. Each plot will have the susceptible population, the total infected population in that demographic group (sums the infected population over the different infection stages) and the recovered population.
- 3: Prints out separate plots for each demographic group. Each plot will have the relative fraction of the population of each infection stage out of the total infected population in that demographic group.
- 4: Prints out one graph that has the total susceptible population, the total infected population and the total recovered population.

`killimages`: Flag to command MATLAB to kill plots before run is over. Class: Scalar.

- 0: Leaves the plots on the screen. You will need to kill them manually .
- 1: Hitting any key will kill the plots one by one.

`printdata`: Flag to print data to screen. If `printdata` is 0, no data will be printed to screen. If it is 1, the code will print to screen, the total population, the total infected population, the population in each subgroup and their time rate of change. In the display, where n is the number of demographic groups and m is the number of infection stages, $y(1)\ldots y(n)$ will correspond to $S_1 \ldots S_n$; $y(n+1)\ldots y(2n)$ will correspond to $I_{11} \ldots I_{n1}$; $y(2n+1)\ldots y(3n)$ will correspond to $I_{12} \ldots I_{n2}$; and so on, with $y((m \times n)+1)\ldots y((m+1)\times n)$ corresponding to $I_{1m} \ldots I_{nm}$ and $y(((m+1)\times n)+1)\ldots y((m+2)\times n)$ corresponding to $R_1 \ldots R_n$. $yt(i)$ will correspond to the time derivative of $y(i)$. Class: Scalar.

`PlotJac`: Flag that tells the code to plot eigenvalues of the Jacobian (if the disease-free equilibrium exists). If `PlotJac` is set to 1, the code will plot the real part of the eigenvalues of the Jacobian of the equations for the infected group at the disease-free equilibrium. If it is set to 0, these plots will not be printed out. The Jacobian will be calculated, regardless. Class: Scalar.

3.4.2 `XTfile`

This file contains the data for the x and τ endpoints — which determine the number of demographic groups and infection stages. The file needs to be in ASCII format. The file is read using `load`. It should contain one column vector with the first entry specifying the number of x endpoints (ie. one greater than the number of demographic groups desired). If the first entry is n, the next n entries should be the x endpoints and there will be $(n - 1)$ demographic groups. The remaining entries are the τ endpoints. All endpoints should be in strictly ascending order. If the first entry does not give the correct number of x endpoints, there will be an error which may or may not be reported by the code. For example, the `XTfile` containing,

```
4
0
8
25
100
1
2
3
4
5
```

would create 3 demographic groups divided in the intervals (0, 8], (8, 25] and (25, 100]; and 4 infection stages divided into the intervals (1, 2], (2, 3], (3, 4] and (4, 5].

This information is used to create `xtype`, `tautype` and `xtautype` classes for the parameters and the variables. The units of x can be of your choosing or x may be unitless. The units of τ should be time.

3.4.3 `SIfile`

This file is used to create an `xtype` class for the initial susceptible population; and an `xtautype` class for the initial infected population. (An `xtype` class is also created for the initial recovered population but the number of people there is automatically set to zero.) The units for these classes are people.

For `FlagData` set to '`FullMP`', `SIfile` needs to be in ASCII format. The file is read using `load`. The first column is the number of people in the susceptible section of each demographic group. The second column is the the number of people in the first infection stage in each demographic group. The third column is the number of people in the second infection stage in each demographic group and so on.

The first row corresponds to the people in the first demographic group, the second row to the people in the second demographic group and so on.

The number of rows must be the same as the number of demographic groups specified in `XTfile`, while the number of columns must be 1 more than the number of infection stages specified in `XTfile` (the first column contains the susceptible group).

For example, for `SIfile` containing the following information,

```
100  0  5  0  0
500  0  0  3  2
300  1  8  0  0
```

the initial conditions would be

$$\begin{array}{llllll} S_1 = 100 & I_{11} = 0 & I_{12} = 5 & I_{13} = 0 & I_{14} = 0 & R_1 = 0 \\ S_2 = 500 & I_{21} = 0 & I_{22} = 0 & I_{23} = 3 & I_{24} = 2 & R_2 = 0 \\ S_3 = 300 & I_{31} = 1 & I_{32} = 8 & I_{33} = 0 & I_{34} = 0 & R_3 = 0 \end{array}$$

with 3 demographic groups and 4 infection stages.

Note: The number of demographic groups and infection stages specified in `SIfile` must be consistent with the data in `XTfile`.

For `FlagData` set to '`FullEP`', `SIfile` needs to be in the same format as the '`FullMP`', but the input data is in endpoint format (so there will be 1 extra row and 1 extra column). The actual interpolated values will also depend on the endpoints specified in `XTfile`.

For `FlagData` set to '`SparseMP`', `SIfile` needs to be in ASCII format. It is read using `fscanf`. The first n entries, where n is the number of demographic groups specified in `XTfile`, need to be the initial susceptible population in each demographic group. The next entries for the number of infected people in each infected subgroup should come in groups of 3. Out of each triplet, the first entry is the demographic group, the second entry is the infection stage and the third entry is the number of infected people. The number of infected people in all demographic groups and infection stages that are not specified by this file are set to 0.

For example, for `SIfile` containing,

```
100
500
300
  1  2  5
  2  3  3
  2  4  2
  3  1  1
  3  2  8
```

the initial conditions would be

$$\begin{array}{llllll} S_1 = 100 & I_{11} = 0 & I_{12} = 5 & I_{13} = 0 & I_{14} = 0 & R_1 = 0 \\ S_2 = 500 & I_{21} = 0 & I_{22} = 0 & I_{23} = 3 & I_{24} = 2 & R_2 = 0 \\ S_3 = 300 & I_{31} = 1 & I_{32} = 8 & I_{33} = 0 & I_{34} = 0 & R_3 = 0 \end{array}$$

with 3 demographic groups and 4 infection stages.

Note: The number of demographic groups and infection stages specified in `SIfile` must be consistent with the data in `XTfile`.

For `FlagData` set to '`SparseEP`', the file needs to be in the same format as the '`SparseMP`', but the input data is in endpoint format. There will need to be one more entry for the susceptible population. The entries for the infected population will also be for the endpoint values so the actual interpolated midpoint values will also depend on the endpoints specified in `XTfile`.

3.4.4 `MMfile`

This file determines the parameters that describe the mixing between the different demographic groups (either q_{ik} or p_{ik} and c_i). The format of this file is determined by `MMflag`.

If `MMflag` is set to '`QCData`', then the code directly reads in the q and c data, calculates p and proceeds as described in §1.1.4. This file is read using the `load` command, and should be in ASCII format. It should contain n rows and $(n+1)$ columns. The first column corresponds to c while the remaining n columns correspond to q. The file should be of the format:

$$\begin{array}{lllll} c_1 & q_{11} & q_{12} & \cdots & q_{1n} \\ c_2 & q_{21} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_n & q_{n1} & q_{n2} & \cdots & q_{nn}. \end{array}$$

Note: The number of demographic groups in this file must be equal to the number specified in `XTfile`.

If `MMflag` is set to '`MixingMatrix`', the code reads in the total number of contacts between the different groups per unit time and then calculates c and p. This file is read using the `load` command and should be in ASCII format. The file should contain an $n \times n$ **symmetric** matrix where n is the number of demographic groups (should be equal to the number defined in `XTfile`). This matrix should contain the **total** number of contacts (by all people) between the different demographic groups, per unit time. The units of M are (Number of Contacts)/Time. The unit of time here must be consistent with the unit of time used in `sp.in` for the other parameters that have time in their dimension.

This option allows for easy interfacing between this code and TRANSIMS[6]. Demographic data can be collected from TRANSIMS and stored in a file in the above format which can then be read by the code. However, care must be taken to ensure that the number of people used to create the mixing matrix must be the same as the total initial population. Otherwise, the average number of contacts per person that this code calculates will be inconsistent with the data in the mixing matrix.

An example of a valid `MMfile` for 3 demographic groups with `MMflag` = '`MixingMatrix`' is

```
67   45   12
45  377  128
12  128  35.
```

Here the people in demographic group 1 have a total of 67 contacts within their own group, 45 contacts with the people in demographic group 2 and 12 contacts with the people in demographic group 3, per unit time. The people

[6]An agent-based model of a virtual city that describes movement and interaction of people.

in demographic group 2 have a total of 45 contacts with group 1, 377 contacts amongst themselves and 128 contacts with group 3, per unit time. The people in group 3 have a total of 12 contacts with group 1, 128 contacts with people in group 2 and 35 amongst themselves, per unit time.

We describe the calculation of the parameters, p and c below.

Let M_{ik} be the total number of contacts between group i and group k per unit time, read from the input file. (This is the same quantity as T_{ik} in §1.1.4 but we use M here to denote an input read from a file.)

We define c_i, the preferred number of contacts per unit time of one person in group i, as the total number of contacts made by group i divided by the total initial population of group i in (14).

$$c_i = \frac{\sum_{k=1}^{n} M_{ik}}{N_i(0)}. \tag{14}$$

We then define the preference for a contact between group i and group k using the definition for the total contacts (6) with the initial population data in (15).

$$p_{ik} = M_{ik} \frac{\sum_{l=1}^{n} c_l N_l(0)}{c_i N_i(0)\ c_k N_k(0)}. \tag{15}$$

Note that now, p_{ik} is no longer restricted to being less than or equal to 1 and must be interpreted as a preference, not a probability. We do not solve for q_{ik}. These parameters no longer have the same interpretation as in [1], but this method serves as an efficient and consistent way of scaling the contacts between the demographic groups, while satisfying the balance constraints.

If `MMflag` is set to'`MixingMatrixPP`', the code reads in the average number of contacts per person per unit time between the different groups and then calculates c and p. This file is read using the `load` command and should be in ASCII format. The file should contain an $n \times n$ matrix (no longer needs to be symmetric) where n is the number of demographic groups (should be equal to the number defined in `XTfile`). This matrix should contain the average number of contacts **per person** between the different demographic groups, per unit time. The units of M are (Number of Contacts)/(Person·Time). The unit of time here must be consistent with the unit of time used in `sp.in` for the other parameters that have time in their dimension.

Note: Although this matrix does not need to be symmetric, the matrix of the total number of contacts between the different demographic groups does need to be symmetric (because the total number of contacts between group i and group k must be equal to the total number of contacts between group k and group i). Thus, when each row of the matrix in `MMfile` is multiplied by the total population of that group, the resulting matrix must be symmetric.

An example of a valid `MMfile` for 3 demographic groups with population sizes $N_1(0) = 100$, $N_2(0) = 50$ and $N_3(0) = 200$ for `MMflag` = '`MixingMatrixPP`' is

```
 7   5   2
10   7   8
 1   2   5.
```

Here, each person in group 1 has, on average, 7 contacts with other people in group 1, 5 contacts with people in group 2 and 2 contacts with people in group 3. Similarly, each person in group 2 has, on average, 10 contacts with people in group 1, 7 contacts with other people in group 2 and 8 contacts with people in group 3. And each person in group 3 has, on average, 1 contacts with people in group 1, 2 contacts with people in group 2 and 5 contacts with other people in group 3. The total mixing matrix for this input file is

$$M = \begin{pmatrix} 700 & 500 & 200 \\ 500 & 350 & 400 \\ 200 & 400 & 1000 \end{pmatrix}.$$

The code calculates this total mixing matrix and then calculates c and p as it would if `MMflag` were set to '`MixingMatrix`'.
Note: If the calculated total mixing matrix is not symmetric, the code will return an error and stop.

3.5 Algorithm

The ODE's are integrated using `ode45`. This method uses variable order Runge-Kutta (4 and 5) with a relative tolerance of 10^{-6} and an absolute tolerance of 10^{-11}. These may be varied, if desired, by changing the values of `Eps` and `AbsEps`, respectively, in `dsdisp.m`.

3.6 Output

The code has two forms of output:

- Some results and plots can be printed to the screen.
- Data is saved to files in the `output` directory.

3.6.1 Displayed Results

There are numerous flags in `sp.in` that allow you to print various results to the screen. We will describe these in some detail below.

If `printdata` is set to 1, time, the populations of the different subgroups, their rate of change, the total population and the total infected population, at intervals of `dtsave`, will be displayed to the screen. On the screen, $y(1)\dots y(n)$ correspond to $S_1\dots S_n$; $y(n+1)\dots y(2n)$ correspond to $I_{11}\dots I_{n1}$; $y(2n+1)\dots y(3n)$ correspond to $I_{12}\dots I_{n2}$; and so on, with $y((m\times n)+1)\dots y((m+1)\times n)$ corresponding to $I_{1m}\dots I_{nm}$ and $y(((m+1)\times n)+1)\dots y((m+2)\times n)$ corresponding to $R_1\dots R_n$. The terms $yt(i)$ correspond to the time derivative of $y(i)$. Although this instant display of the results may be useful when setting up parameters, it will slow the code down.

Depending on the value of `plotfigures`, different plots may be displayed as shown in Table 3. There is currently no option to show the relative impact on disease spread of each infectious subgroup. We have yet to decide on the final form of the expression for the relative impact, but we hope to add it to the model soon. The data is plotted from time equals 0 (after re-initialization — see §3.3 for more details) to the time specified in `tfinal`. The resolution of the plots is controlled by `dtsave`. The smaller `dtsave` is, the better the resolution. The plots are not saved by the code. If you do wish to save them, you will need to save them manually.

If `killimages` is set to 1, the code will wait for user-input to kill each figure. (Depending on the value of `plotfigures` there may be as many figures as demographic groups.) The figures will be killed sequentially from the first demographic group to the last. If you wish to view the plots while continuing to use MATLAB, set `killimages` to 0. You will then need to kill the figures manually.

0:	Prints no plots.
1:	Prints out separate plots for each demographic group. Each plot will show the susceptible population, the population in each infected stage, and the recovered population for that demographic group. Each plot has different line colors for the susceptible, infected and recovered population, but the same colour for the different infected subgroups; and the same linestyle for all curves. Also, there is no legend on these plots. The labels of the different subgroups are shown using the `text` command. The label is placed slightly above the maximum of that curve — so if there are a number of monotonically increasing curves, all their text labels will be placed on the right of the figure. This layout may not necessarily be visually pleasing. If you wish to show these data in a professional setting, I would recommend writing your own script to plot the data. You may copy and paste code from `plotfig.m`, if it helps.
2:	Prints out separate plots for each demographic group. Each plot will have the susceptible population, the total infected population in that demographic group and the recovered population in that group. The infected population is summed over the different infection stages for each demographic group. These plots do have different linestyles and line colors. They have a legend and the lines are drawn 2pt thick.
3:	Prints out separate plots for each demographic group . Each plot will have the relative fraction of the population of each infection stage out of the total infected population in that demographic group. Again, each plot has the same line colors and linestyles. There is no legend and the curves are labeled by text which appears slightly above the maximum value of that curve. It may again be necessary to write your own plotting routines to display these graphs in a more visually pleasing manner.
4:	Prints out one graph that has the total susceptible population, the total infected population and the total recovered population of the whole system. This plot does have different linestyles and colors. There is a legend and the lines are drawn 2pt thick.

Table 3: The different options for plotting data for values of `plotfigures`

If `plotJac` is set to 1, the real part of the eigenvalues of the Jacobian at the disease-free equilibrium will be plotted on the screen. The x-axis is simply a labelling of the eigenvalues while the y-axis is the real part of the eigenvalues. The different eigenvalues are shown by crosses.

3.6.2 Saved Results

There are currently three files saved by the code. They contain the population of each subgroup at different times, the time to re-initialization and the Jacobian of the linearized system around the disease-free equilibrium. The files saved are:

- `ytData.dat`
- `Time_Init.dat`
- `JacTE.dat`

These are all saved in the `output` directory. If you wish to use different names for these files you can change the values of `ytfile`, `tinitfile` and `jacfile`, respectively, in `dsdisp.m`.

1. `ytData.dat`:
   This file contains the values of time, the population of the subgroups and their rates of change, at intervals of `dtsave`, from time $t = 0$ (after re-initialization) to `tfinal`. These are stored in ASCII format in `ytData.dat` using the subroutine `saveyt.m`. The first column of `ytData.dat` contains the values of time. Values in columns 2 through $(n + 1)$ contain data for groups S_1 through S_n. The next n columns contain data for subgroups I_{11} through I_{n1}. The n columns after that contain data for I_{12} through I_{n2} and so on. The last n columns contain data for R_1 through R_n. The first row corresponds to time 0. The second row to time `dtsave`, the third row to time 2×`dtsave` and so on.

   You may use `loadsirp.m` to retrieve the data from `ytData.dat`. The subroutine `loadsirp.m` takes in 4 input parameters. The first parameter is the filename where the data is stored. The second parameter is a flag that determines the format of this file. This can be either 'ASCII' or 'MAT'. The third parameter is the number of demographic groups and the fourth parameter is the number of infection stages in the model. For retrieving data saved by `dsdisp.m`, use '`ytData.dat`' for the first parameter, 'ASCII' for the second parameter and then enter the number of demographic groups and infection stages for the third and fourth parameters.

   The function `loadsirp.m` has four output parameters. The first parameter, `T` is a column vector of time values from 0 to `tfinal` at intervals of `dtsave`. The second parameter, `SP`, is a matrix containing the information on the susceptible population. The first column corresponds to time $t = 0$; the second to time $t =$`dtsave` and so on — with each column corresponding to a value of time defined in `T`. The first row contains the values of S_1 at each of these times, the second row the values of S_2 and so on, with the last row containing the values of S_n. The third output parameter, `IP`, contains the infected population values. This is a 3D array where each page corresponds to time defined in `T`. Each row corresponds to the demographic group and each column corresponds to the infection stage. The fourth output parameter, `RP` is the recovered population and has a similar structure to `SP`.

Typing

```
> help loadsirp
```

in MATLAB in the directory `dsmodel/code` will also produce this information on the format of the output.

2. `Time_Init.dat`:
   This file contains the time at which the code re-initializes the population as described in §1.1.6. The file is saved in ASCII format and contains only a scalar.

3. `JacTE.dat`:
   This file contains the Jacobian of the system (of only the infecteds) at the disease-free equilibrium. The Jacobian is calculated as described in §1.1.5. This file is also stored in ASCII format. It contains one matrix with all the terms of the Jacobian.

3.7 Sample Run

To run this code, open MATLAB and enter the directory, `dsmodel/code`. Then, type in

```
> dsdisp.
```

For sample runs, some input files have already been created. In the directory, `dsmodel/input` there are three subdirectories:

- `AIDS4G`
- `AIDSDISP`
- `AIDSSP`.

To run any of these models, copy all four input files from that subdirectory into `dsmodel/input` and from `dsmodel/code` in MATLAB run `dsdisp`. This will run the code with the parameters set up for that model.

To run the code with your own parameters, it may also help to start with `sp.in` from one of these models and change that as you wish to keep the format of that file.

3.8 User-Defined Equations

To allow the user to expand this model to include more dynamics, such as behaviour change or migration between demographic groups, we have included functions that can be changed by the user to modify the system of equations.

The system variables, S, I and R are stored as one large column vector, y. The format of y is the same as that in §3.6.1. The first n terms, $y(1) \dots y(n)$, correspond to $S_1 \dots S_n$. The next n terms, $y(n+1) \dots y(2n)$, correspond to $I_{11} \dots I_{n1}$; $y(2n+1) \dots y(3n)$ correspond to $I_{12} \dots I_{n2}$; and so on, with $y((m \times n)+1) \dots y((m+1) \times n)$ corresponding to $I_{1m} \dots I_{nm}$ and $y(((m+1) \times n)+1) \dots y((m+2) \times n)$ corresponding to $R_1 \dots R_n$.

To change the derivatives, you may add or subtract terms in `fuser.m`. This subroutine returns a column vector (with the same format as y) that is added to the system of equations (1). Let us call the output vector of `fuser.m`, u. Adding a term to $u(1)$ will add that term to $\dot{S_1}$; adding it to $u(n)$ will add it to $\dot{S_n}$; $u(n+1)$ to $\dot{I_{11}}$ and so on.

For example, to convert the model to an SIRS model (16), with a recovery rate, ρ, `fuser.m` would need to return the vector (17). However, the user would need to input the new parameter, ρ, directly in `fuser.m`

$$\begin{aligned}
\frac{dS_i}{dt} &= \Lambda_i - (\mu_{iS} + \lambda_i(t))S_i + \rho_i R_i && \text{(16a)}\\
\frac{dI_{i1}}{dt} &= \lambda_i(t)S_i - (\gamma_{i1} + \mu_{i1})I_{i1} && \text{(16b)}\\
\frac{dI_{ij}}{dt} &= \gamma_{i,j-1}I_{i,j-1} - (\gamma_{ij} + \mu_{ij})I_{ij} \qquad \text{for } 2 \le j \le m && \text{(16c)}\\
\frac{dR_i}{dt} &= \gamma_{im}I_{im} - (\rho_i + \mu_{iR})R_i && \text{(16d)}
\end{aligned}$$

for $1 \le i \le n$.

$$u = \begin{pmatrix} \rho_1 R_1 \\ \rho_2 R_2 \\ \vdots \\ \rho_n R_n \\ 0 \\ \vdots \\ 0 \\ \vdots \\ \vdots \\ 0 \\ \vdots \\ 0 \\ -\rho_1 R_1 \\ -\rho_2 R_2 \\ \vdots \\ -\rho_n R_n \end{pmatrix} \tag{17}$$

Also, any changes to `fuser.m`, will require corresponding changes to `jacuserTE.m` to keep the equations consistent. The output of `jacuserTE.m` is a matrix, following the format of §1.1.5. This output is added to the jacobian calculated in `getjacobianTE.m`. In the above example, as no changes are made to the equations for I, the jacobian is unchanged.

3.9 Killing Execution

To kill execution at any time, type in Ctrl + c. As some data is saved into output files at the end of the run, in most cases, not all the data will be saved. Some variables in MATLAB's working memory may be changed.

4 Bugs and Modifications

Feel free to contact us:

- `nakul@math.arizona.edu`
- `restrepo@math.arizona.edu`.